CS3025 Compiladores

Tarea de Programación 1

Salida: 25/08/2023 Entrega: 1/Septiembre/2023 5pm

Implementar un analizador léxico (scanner) para el lenguaje de la máquina de pila SM (definido en la siguiente sección). La implementación del scanner tendrá que estar basada en los estados y transiciones del autómata finito presentado con las instrucciones abajo (Figura 1): el scanner deberá implementar explícitamente este autómata.

1. Sintaxis de SM

El lenguaje de máquina de pila SML está definido por la siguiente gramática:

<program> ::= <instruction>+
<instruction> ::= Label? ((<jmpinstr> <id>) | <unaryinstr> | (**push** <num>) <eol>
<jmpinstr> ::= **jmpeq** | **jmpgt** | **jmpge** | **jmplt** | **jmple** | **goto**
<unaryinstr> ::= **skip** | **pop** | **dup** | **swap** | **add** | **sub** | **mul** | **div** | **store** | **load**

La especificación léxica está dada por los siguientes patrones (expresiones regulares)

digito ::= [0-9]
carácter :: = [a-zA-Z]
<palabra-reservada> ::= **push** | **jmpeq** | **jmpgt** | **jmpge** | **jmplt** | **jmple** | **skip** | **pop** | **dup** | **swap** | **add** | **sub** | **mul** | **div** | **store** | **load**
<id> ::= carácter | (caracter | digito | '_')*
<label> ::= <id> :
<num> ::= digito+
<eol> ::= '\n' +
<ws> ::= (' ' | '\t')+

Las unidades léxicas <id>, <num>y <eol> , y todas las palabras reservadas, generan un token. El patrón <ws> no genera token; denota los espacios en blancos -y deberá ignorarse. Los tokens llevaran el nombre de sus unidades léxicas pero en mayúscula e.g. el token NUM denota la unidad lexica <num>. Solo los tokens correspondientes a <num> y <id> tendrán como atributo al lexema. Resumiendo, los tokens a generar son:

- Un tipo de token por cada palabra reservada, con el mismo nombre del lexema. Por ejemplo, la palabra reservada **pop** generara el token POP.
  Asi tendremos PUSH, JMPEQ, JMPGT, …, STORE, LOAD
- NUM(lexema) lexema es <num>
- ID(lexema) lexema es <id>
- LABEL(id) donde el lexema es id:
- EOL

Además se generara un token especial para reportar errores:

- ERR(lexema) donde el lexema será el carácter invalido.
- Los espacios en blanco (<ws>) serán igorados i.e. no se generara ningún token.

En nuestro programa, los tipos de token serán definidos por `enum Type`:

enum TokenType { ID, NUM, EOL, PUSH, JMPEQ, …, STORE, LOAD, ERR, END }

2. El Autómata

El autómata que acepta las cadenas validas en nuestro lenguaje, es decir, las cadenas que pertenecen al lenguaje definido por las expresiones regulares en la especificación léxica, está definido en la Figura 1. Este autómata será usado para la implementación del scanner.

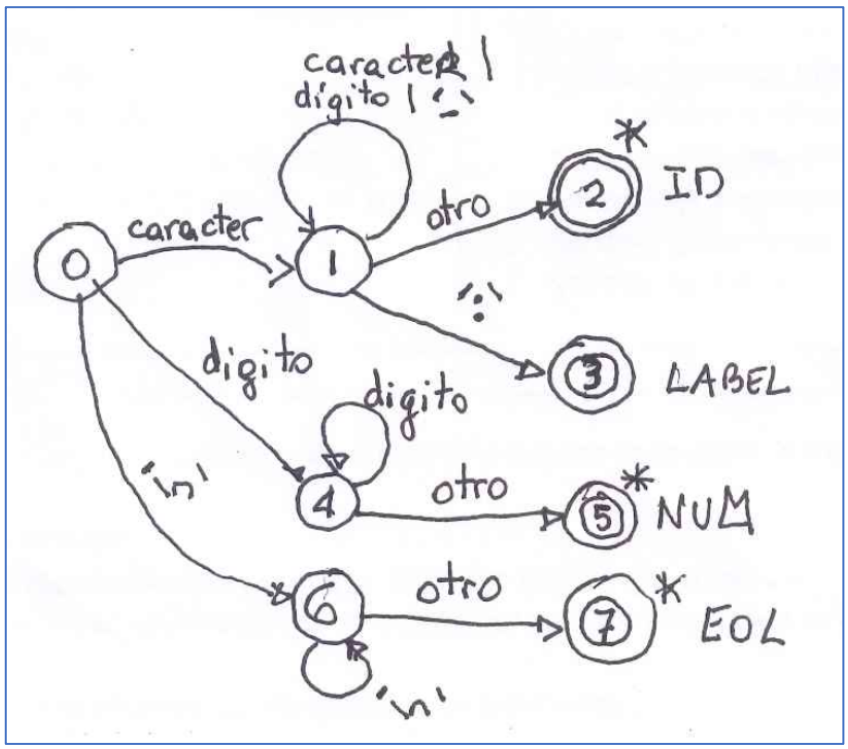

Figura 1

Nótese lo siguiente:

- el uso de asterisco en algunos estados finales, indicando que un `rollBack` es necesario.
- La falta de transiciones para las palabras reservadas se debe a que toda palabra reservada deberá ser aceptada primero como ID. Una vez que se acepta un ID se deberá verificar primero si el lexema pertenece a la lista de palabras reservadas. Si es así, se genera el token respectivo para la palabra reservada; de lo contrario, se genera el token ID.

3. El Scanner

El archivo `scanner_sm.cpp` contiene el código esqueleto del scanner para SML. El archivo contiene dos clases `Token` y `Scanner`, a cargo del análisis lexicográfico (lexical analysis), y una función `main`.
Tomar en cuenta lo siguiente:

- El programa tomara un parámetro de la línea de comandos: el nombre del archivo a analizar.
  Para esto se deberá cambiar el programa para que abra y lea el input desde este archivo. Es valido leer todo el archivo a un string y pasarlo al constructor del scanner e.g. como se hace en `svm.cpp` (lab1).
- Las palabras reservadas deberán guardarse en una estructura de datos de busqueda rápida. El código que genera los tokens correspondientes a las palabras reservadas deberá inspeccionar esta estructura para verificar si el ID que se encontró es una palabra reservada o no.
- Ojo: Se deberá modificar el operator overload de `operator<<`.

4. Ejemplo input y output

Tomemos como ejemplo al archivo `ejemplo1.sm`. Una vez compilado nuestro programa:

```
>> g++ scanner_sm.cpp
```

Y ejecutado con el archivo ejemplo

```
>> ./a.out ejemplo1.sm
```

Se deberá obtener lo siguiente en standard output

```
PUSH
NUM(3)
EOL
LABEL(L1)
JMPGT
EOL
MUL
EOL
ADD
EOL
GOTO
ID(L2)
EOL
```

5. Que entregar

El archivo `apellido1_apellido2_scanner_sm.cpp`

Donde `apellido1` y `apellido2` corresponden a los apellidos de los miembros del grupo.